

Nombre y apellido:

HERNAN GIANNI

L.U.:

Nº 11

Turno:

NOCHE

## Algoritmos y Estructuras de Datos II

### Segundo parcial – 2<sup>do</sup> cuatrimestre 2018

- El parcial es a libro abierto.
- Cada ejercicio debe entregarse en hojas separadas.
- Incluir en cada hoja el número de orden asignado, número de hoja, LU, turno, apellido y nombre.
- Antes de entregar, remover los "pelitos" del borde de las hojas, si hubiere.
- Cada ejercicio se calificará con Perfecto, Aprobado, Regular, o Insuficiente.
- El parcial está aprobado si el ejercicio 1 tiene al menos A, y entre los ejercicios 2 y 3 hay al menos una A.

A	A	A
---	---	---

Perfect

### Ej. 1. Diseño

(ALEXIS)

El sistema MULT.AR se encarga de registrar las multas de los vehículos a lo largo y ancho del país. Cada *localidad* posee un conjunto de *cámaras* y tiene un conjunto de *vehículos* registrados (aunque los vehículos pueden circular libremente por todo el país). Cuando un vehículo sobrepasa la velocidad máxima y es registrado por una cámara, se emite una *multa* al infractor. Por lo tanto para registrar una multa se necesita el código identificador de la cámara, la patente del vehículo que cometió la infracción y el monto de la misma. En cualquier momento se pueden abonar las multas de un vehículo, pero en ese momento se deben abonar todas las multas existentes del vehículo. Esta acción elimina todos los registros del sistema asociados a esas multas.

#### TAD MULT.AR

##### observadores básicos

localidades	: mar	→ conj(loc)	
camarasDe	: mar $m \times \text{loc } \ell$	→ conj(camara)	$\{\ell \in \text{localidades}(m)\}$
vehiculosDe	: mar $m \times \text{loc } \ell$	→ conj(vehiculo)	$\{\ell \in \text{localidades}(m)\}$
multasPorV	: mar $m \times \text{vehiculo } v$	→ multiconj(multa)	$\{v \in \text{vehiculos}(m)\}$

##### generadores

iniciar	: dice(loc $\times$ camara)	→ mar	
registrarVehiculo	: mar $m \times \text{loc } \ell \times \text{vehiculo } v$	→ mar	$\{\ell \in \text{localidades}(m) \wedge v \notin \text{vehiculos}(m)\}$
multar	: mar $m \times \text{vehiculo } v \times \text{camara } c \times \text{nat}$	→ mar	$\{v \in \text{vehiculos}(m) \wedge c \in \text{camaras}(m)\}$
abonar	: mar $m \times \text{vehiculo } v$	→ mar	$\{v \in \text{vehiculos}(m)\}$

##### otras operaciones

camaras	: mar	→ conj(camara)	$\{\}$
vehiculos	: mar	→ conj(vehiculo)	$\{\}$
multasPorLocalidad	: mar $m \times \text{loc } \ell$	→ multiconj(multa)	$\{\ell \in \text{localidades}(m)\}$

##### axiomas

...

Fin TAD

Las *localidades* y los *vehículos* se representan con strings (acotados en el caso de los vehículos, por ser las placas de las patentes), y las *cámaras* se representan con números naturales. Una *multa* se representa con una tupla  $\langle v: \text{vehiculo}, c: \text{camara}, \text{monto}: \text{nat} \rangle$ .

Se debe realizar un diseño que cumpla con los siguientes órdenes de complejidad en el peor caso, siendo  $\ell$  el nombre de la localidad,  $n$  la cantidad de cámaras del sistema y  $m$  la cantidad de multas del vehículo en cuestión:

- Dada una localidad  $\ell$ , obtener los vehículos registrados, las cámaras y las multas (incluyendo las de sus vehículos y las de sus cámaras), cada operación en  $O(|\ell|)$ .
- Dado un vehículo, obtener su localidad y sus multas, ambos en  $O(1)$ .
- Abonar (eliminando del sistema) las multas de un vehículo en  $O(m)$ .
- Dada una cámara, un vehículo y un monto, registrar una nueva multa en  $O(\log n)$ .

- Escriba la estructura de representación del módulo explicando detalladamente qué información se guarda en cada parte y las relaciones entre las partes. Describa también las estructuras de datos subyacentes.
- Escriba el algoritmo para abonar (eliminando del sistema) las multas de un vehículo y justifique el cumplimiento de la complejidad solicitada. Para las demás funciones, justifique en castellano por qué se cumple la complejidad pedida.

## Ej. 2. Ordenamiento

La tienda de ropa y accesorios *Balafella* registra todos los días las ventas realizadas por sus empleados. Al final del día, se tiene una lista de *ventas* en la que cada venta se representa con una tupla  $\langle \text{empleado: string, monto: importe} \rangle$ , donde *empleado* es el nombre de usuario del empleado que realizó la venta y *monto* es el importe en pesos con hasta dos decimales de precisión.

Con el objetivo de premiar a los empleados más activos, se desea ordenar la lista de ventas de forma que aparezcan primero las ventas de los empleados que más ventas realizaron. Es decir, una venta de un empleado que realizó  $x$  ventas debe aparecer antes que una venta de otro que realizó  $y$  ventas, siempre que  $x > y$ . Además, las ventas de un mismo empleado se deben ordenar en forma decreciente según los importes. Si dos (o más) empleados realizaron la misma cantidad de ventas, el orden final de todas estas ventas debe estar dado por los importes de las mismas, en forma decreciente. Por ejemplo, si tenemos la siguiente lista de ventas:

$\langle \text{pedro, \$90.50} \rangle, \langle \text{juan, \$5.80} \rangle, \langle \text{juan, \$15.00} \rangle, \langle \text{ana, \$85.50} \rangle, \langle \text{juan, \$30.25} \rangle, \langle \text{ana, \$21.30} \rangle$

el resultado debería ser

$\langle \text{juan, \$30.25} \rangle, \langle \text{juan, \$15.00} \rangle, \langle \text{juan, \$5.80} \rangle, \langle \text{ana, \$85.50} \rangle, \langle \text{ana, \$21.30} \rangle, \langle \text{pedro, \$90.50} \rangle$

Si a la lista anterior le agregáramos la venta  $\langle \text{ana, \$18.00} \rangle$ , haciendo que *ana* empate en ventas con *juan*, entonces el resultado debería ser

$\langle \text{ana, \$85.50} \rangle, \langle \text{juan, \$30.25} \rangle, \langle \text{ana, \$21.30} \rangle, \langle \text{ana, \$18.00} \rangle, \langle \text{juan, \$15.00} \rangle, \langle \text{juan, \$5.80} \rangle, \langle \text{pedro, \$90.50} \rangle$

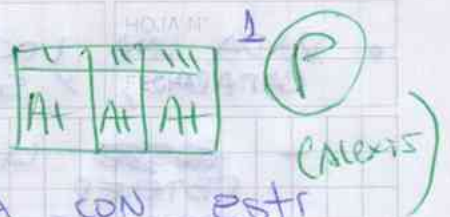
Se pide escribir un algoritmo que tome un arreglo de ventas y la ordene de la forma descripta. Sabiendo que los importes de las ventas son siempre menores a \$100000 y que los nombres de usuario tienen longitud acotada, se pretende que el algoritmo propuesto tenga complejidad  $O(n)$ , donde  $n$  es la cantidad de ventas del arreglo. Se espera que el algoritmo no abuse del uso de la memoria extra más de lo necesario. Más precisamente, no se admiten soluciones que utilicen arreglos (o vectores) de 100000 o más posiciones.

## Ej. 3. Dividir y Conquistar

Decimos que un árbol binario de booleanos está *equilibrado* si para todo nodo del árbol, ocurre que la cantidad de *falsos* en su subárbol izquierdo difiere en a lo sumo uno con la cantidad de *verdaderos* en su subárbol derecho.

Sabiendo que contamos con las operaciones de árbol binario *nul?*, *raiz*, *izq* y *der*, cuyas complejidades temporales son  $O(1)$ , proponer un algoritmo que utilice la técnica de *Dividir y Conquistar* para determinar si un árbol binario de booleanos está equilibrado. El algoritmo propuesto debe tener complejidad  $O(n)$ , donde  $n$  es la cantidad de nodos del árbol, sin importar si el mismo está o no balanceado (notar que estar equilibrado no implica estar balanceado ni viceversa). La complejidad del algoritmo debe estar correctamente justificada.





1) EL TAD MULT. AR SE REPRESENTA CON ESTR  
DONDE ESTR ES:

tupla  $\langle \checkmark$  localidades: diccTrie (localidad,  $\langle$  conjTrie (vehiculo),  
conjLineal (camara),  
MultiConjLineal (multa)  $\rangle$ ,

$\checkmark$  vehiculos: diccTrie (vehiculo,  $\langle$  it a diccTrie +  
(localidad,  $\langle \dots \rangle$ )  
, conjLineal (it a MultiConjLineal  
(multa))  $\rangle$ ,

$\checkmark$  camaras: diccAVL (camara, it a diccTrie (localidad,  
 $\langle$  conjTrie (vehiculo)  $\dots \rangle$ )

$\rangle$

- EN LOCALIDADES TENGO UN DICCIONARIO QUE DADA UNA LOCALIDAD ME DA COMO SIGNIFICADO UNA TUPLA CON SUS VEHICULOS, SUS CAMARAS Y SUS MULTAS. ESTA IMPLEMENTADO SOBRE TRIE PARA PODER BUSCAR UNA LOCALIDAD EN  $O(|I|)$ . EL CONJUNTO DE VEHICULOS TAMBIEN PARA QUE BUSCAR UNO SEA  $O(1)$  (IE ES UTIL PARA REGISTRAR MULTA)

- ~~VEHICULOS~~ VEHICULOS ES UN DICCIONARIO IMPLEMENTADO SOBRE TRIE. AL SER LAS CLAVES ACOTADAS PUEDO BUSCAR, AGREGAR Y Borrar UNA EN  $O(1)$ . EN SU SIGNIFICADO GUARDO, EN SU SIGNIFICADO GUARDO UN ITERADOR A LA LOCALIDAD DE ESE VEHICULO Y UN CONJUNTO DE ITERADORES A MULTI CONJ DE MULTA. SON ITERADORES PARA PODER BORRARLAS EN  $O(1)$  Y ESTE CONJUNTO ES LINEAL PARA PODER AGREGAR UN ITERADOR EN  $O(1)$

- CAMARAS ES UN DICCIONARIO IMPLEMENTADO SOBRE AVL LO QUE ME PERMITE BUSCAR, AGREGAR Y BORRAR EN  $O(\log n)$ . EN SU SIGNIFICADO TENGO UN ITERADOR A SU LOCALIDAD YA QUE VOY A NECESITAR EN REGISTRAR MULTA PARA LO SABER CUAL ES LA ~~CAMARA~~ LOCALIDAD DE LA CAMARA QUE TORO LA INFRACCION.



• DADA UNA LOCALIDAD OBTENER SUS VEHÍCULOS, CÁMARA, Y SUS MULTAS EN  $O(|R|)$

- ~~PARA~~ LA LOCALIDAD EN e.localidades OBTENER DEVUELVE LOS TRES CONJUNTOS EN  $O(|R|)$

✓ YA QUE ES UNA BUSQUEDA EN UN DICCIONARIO IMPLEMENTADO SOBRE TRIE

• DADO UN VEHÍCULO, OBTENER SU LOCALIDAD Y SUS MULTAS  $O(1)$

- HACIENDO OBTENER DEL VEHÍCULO EN CUESTIÓN EN e.vehículos ME DEVUELVE UNA TUPLA CON UN ITERADOR A SU LOCALIDAD Y UN CONJUNTO DE ITERADORES A SUS MULTAS. ESTA OPERACIÓN ES  $O(1)$  YA QUE LAS CLAVES DEL DICCIONARIO SOBRE TRIE SON ACOTADAS EN ESTE CASO.

• REGISTRAR MULTA  $O(\log n)$

- BUSCO LA CÁMARA QUE REGISTRO LA MULTA EN e.cámaras. TARDA  $O(\log n)$  AL TRATARSE DE UNA BUSQUEDA EN UN AVL.

- USANDO EL SIGNIFICADO DE LA CÁMARA EN e.cámaras ~~DETERMINO~~ EL ACCESO A LA LOCALIDAD DE LA MISMA (e.localidades)  $O(1)$

- EN EL SIGNIFICADO DE LA LOCALIDAD EN e.localidades TENGO SUS VEHÍCULOS. BUSCO SI EL VEHÍCULO AL QUE SE LE HIZO LA MULTA PERTENECE A LA MISMA LOCALIDAD\*. AL TRATARSE DE UN CONJUNTO SOBRE TRIE CON CLAVES ACOTADAS LA BUSQUEDA ES  $O(1)$ . DE SER ASÍ AGREGO LA MULTA AL MULTICONJUNTO DE MULTAS DE LA LOCALIDAD (LO PUEDO HACER EN  $O(1)$  AL SER UN MULTICONJUNTO LINEAL) Y ME GUARDO UN ITERADOR. BUSCO EL VEHÍCULO EN e.vehículos Y GUARDO EN EL CONJUNTO QUE TIENE EN SU SIGNIFICADO EL ITERADOR  $O(1)$ .

✓ SI LA LOCALIDAD DE LA CÁMARA Y EL VEHÍCULO DIFIEREN, REALIZO LO MISMO QUE ANTES PERO LUEGO USO EL ITERADOR A ~~LA~~ LOCALIDAD QUE TIENE EL VEHÍCULO EN SU SIGNIFICADO (e.vehículos) Y AGREGO LA MULTA AL MULTICONJUNTO ALLÍ TAMBIÉN, ME GUARDO UN ITERADOR Y LO AGREGO A LAS MULTAS DEL VEHÍCULO (SIGNIFICADO



DE E.VEHICULOS), TODAS ESTAS OPERACIONES  
SE REALIZAN EN  $O(1)$

registrar multa (in m : multa, in/out, e: estr) {

vehiculo v  $\leftarrow T_1(m)$   $O(1)$

it (conjLineal)<sup>i</sup>  $\leftarrow$   $\text{creaIt}(T_2(\text{obtener}(v, e.vehiculos)))$   $O(1)$

while ( hay siguiente (~~i~~ i )) {  $O(m)$

Borrar siguiente (siguiente(i))  $O(1)$

Avanzar (i)  $O(1)$

}

Borrar  $\left( \text{obtener}(v, e.vehiculos) \right)$   $O(m)$

}

BASICAMENTE LO QUE HACE EL ALGORITMO ES BUSCAR EL VEHICULO AL CUAL SE LE REALIZO LA MULTA EN E.VEHICULOS Y CREAR UN ITERADOR PARA RECORRER EL CONJUNTO DE ITERADORES QUE HAY EN LA SEGUNDA COMPONENTE DE SU SIGNIFICADO. ~~PARA EL ITERADOR QUE TENGO EN EL CONJUNTO ES ELIMINAR LA MULTA EN  $O(1)$  DEL CONJUNTO AL QUE PERTENEZCA. UNA VEZ ELIMINADAS TODAS LAS MULTAS, BORRO EL CONJUNTO.~~ LO QUE HAGO CON

ACLARACION : EL CONJUNTO QUE TIENE UN VEHICULO EN SU SIGNIFICADO EN E.VEHICULOS TIENE A LO SUJO 2m ~~2m~~ ELEMENTOS YA QUE SI LA ~~LOCALIDAD~~ LOCALIDAD DE LA CANADA Y EL VEHICULO ~~SON~~ CUANDO SE REGISTRA UNA DENUNCIA DIFEREN, DE GUARDO AMBOS ITERADORES PARA PODER BORRAR LA MULTA DE LOS 2 CONJUNTOS EN LOS QUE VA A ESTAR (MULTAS DE LA LOCALIDAD DE LA

CARRERA Y RUTAS DE LA LOCALIDAD DEL VEHICULO).

AUN SI ~~LA~~ TUVIERA 20m ITERADORES EN EL  
CONJUNTO RECORRIDO ES 0(m)



2) void ordenarVentas (in a: arreglo (venta))  
out

O(1) d ← crear DicTrie (empleado, < cantVentas  
, lista (venta) >

for (i from 0 to n-1) { O(n)

if (!definido( $\Pi_1(a[i])$ , d))

O(1) lista l ← crear

O(1) definir ( $\Pi_1(a[i])$ , < 1,  $\text{ag}(l, a[i])$  >)

else

O(1)  $\Pi_1(\text{significado}(a[i], d))++$ ;

O(1)  $\text{ag}(\Pi_2(\text{significado}(a[i], d), a[i])$   
Otras

O(n) buckets ← crear Arreglo de n listas

O(1) it ← crear IT DicTrie

O(n) while (haySiguiente(it))

$\text{agOtras}(\text{buckets}[\Pi_1(\text{significado}(\text{siguiente}(it), d))]$   
/  $\Pi_2(\text{significado}(\text{siguiente}(it), d))$ )

O(1) Avanzar (it)

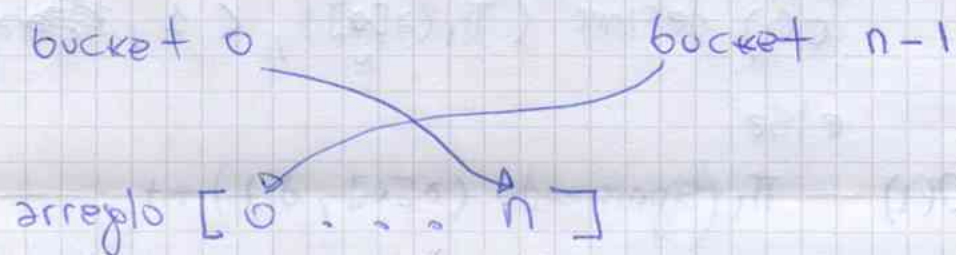
~~for~~  
for ( $i = 0 \dots n-1$ ) {  $O(\sum_{i=0}^{n-1} b) = O(n)$

ordenar  $B[i]$  usando RadixSort  $O(b)$   
en orden de mayor a menor

~~crear~~ listas de buckets  $O(n)$

copiar  
en orden ~~al~~ arreglo ~~en~~  
contrario

osea ~~de~~ de  $n-1$  a  $0$   
~~menor~~



Excelente!



2) - creo un diccionario (empleado, <contventas, lista(venta)>)  
vario en  $O(1)$   
arreglo

- recorro el ~~arreglo~~ y agrego al empleado al trie si no esta y voy sumando uno por cada venta suya que encuentre y agregando a la lista en  $O(1)$  ya que los nombres de usuarios tienen longitud acotada, y agrego a las listas

- hago un bucket sort: recorro las claves del trie en  $O(n)$  y de acuerdo a su contventas - 1, ubico en uno de los  $n$  buckets su lista de ventas

$$O(n) + O(\text{recorrer toda las listas que a } \text{es el tamaño del arreglo inicial}) \\ = O(n)$$

- como los numeros estan en base 10 y son menores que 100000 se que a lo sumo tienen 5 digitos, por lo tanto puedo usar radixSort y ordenar las ~~ventas~~ en  $O(5 \cdot b)$  donde  $b$  es la cant de ventas que estan en la lista \*

aclaraciones: - son  $n$  buckets por que a lo sumo un empleado ~~tuvo~~  $n$  ventas

- cuando como una lista de ventas de un empleado a un bucket lo que hago es ir ~~agregando~~ agregando atras los elementos a la lista que ya estaba en el bucket

- \* ~~ordenar~~ ordenar con radix cada una de las  $n$  listas me lleva en total  $O(n)$

ya que cada ~~de~~ los ~~de~~ ventas  
sumadas de todos los buckets son  
n

$$O\left(\sum_{i=0}^{n-1} b\right) = O(n)$$

contrario para que  
quede de maior a menor

- luego copio en orden las listas de cada bucket al ~~array~~ arreglo  $O(n)$



3)  $\langle \text{bool}, \text{nat}, \text{nat} \rangle$  <sup>Aux</sup>  $\text{estEquilibrado?} (\text{in } a: \text{arbol}(\text{bool}))$

if (nil? (a))  
 return  $\langle \text{true}, 0, 0 \rangle$   $O(1)$

else  
~~if (nil? (der(a)))~~  
~~if (nil? (izq(a)))~~  
 der  $\leftarrow$  ~~estEquilibrado?~~ (der(a))  $T(n/2)$   
 izq  $\leftarrow$  ~~estEquilibrado?~~ (izq(a))  $T(n/2)$   
 if ( $\Pi_1(\text{der})$  or  $\Pi_1(\text{izq})$ )  
 return  $\langle \text{false}, 0, 0 \rangle$   $O(1)$

else  
 if ( $\Pi_3(\text{izq}) + 1 == \Pi_2(\text{der})$   
 or  $\Pi_3(\text{izq}) - 1 == \Pi_2(\text{der})$   
 or  $\Pi_3(\text{izq}) == \Pi_2(\text{der})$ )  $O(1)$   
~~if (raiz(a) == true)~~  
 return  $\langle$  ~~true~~  $, 1 + \Pi_2(\text{izq}) + \Pi_2(\text{der})$   
 $, \Pi_3(\text{izq}) + \Pi_3(\text{der}) \rangle$   $O(1)$

else  
 return  $\langle \text{true}, \Pi_2(\text{izq}) + \Pi_2(\text{der}),$   
 $\Pi_3(\text{izq}) + \Pi_3(\text{der}) + 1 \rangle$   $O(1)$

else  
 return  $\langle \text{false}, 0, 0 \rangle$   $O(1)$

SI ESTA BALANCEADO

$$T(n) = 2T(n/2) + O(1)$$

$$T(n) \in O(n)$$

SI EL ARBOL NO ESTA BALANCEADO

$$T(n) \in O(n) \iff \exists c, n_0 \mid n \geq n_0 \\ T(n) \leq c \cdot n$$

$$T(n) = T(izq) + T(der) + O(1)$$

$$\leq c_{izq} \cdot n_{izq} + c_{der} \cdot n_{der}$$

$$\leq \max(c_{izq}, c_{der}) \cdot n_{izq} + \max(c_{izq}, c_{der}) \cdot n_{der}$$

$$= \max(c_{izq}, c_{der}) \cdot (n_{izq} + n_{der})$$

$$= \max(c_{izq}, c_{der}) \cdot (n - 1)$$

$$\leq \underbrace{\max(c_{izq}, c_{der})}_{\text{LO TONO}} \cdot n$$

LO TONO  
COMO  
CONSTANTE

QUEDA  
PROBADO

$$T(n) \in O(n)$$



```

bool estaEquilibrado (in a: arbol (bool)) {
    return #T1 (estaEquilibradoAux(a)) 0(n)
}

```

LA IDEA DEL ALGORITMO ES LA SIGUIENTE:

~~PRIMERO DEVUELVO SI EL ARBOL ES~~

- VOY A TENER UNA FUNCION `estaEquilibradoAux` QUE VA DEVOLVER UNA TUPLA DONDE LA PRIMER COMPONENTE SIGNIFICA SI EL ARBOL ESTA EQUILIBRADO O NO, LA SEGUNDA LA CANTIDAD DE VALORES VERDADEROS EN EL ARBOL Y LA TERCERA LA CANTIDAD DE VALORES FALSOS.
- SI EL ARBOL ES NIL DEVUELVO  $\langle \text{TRUE}, 0, 0 \rangle$  PUES ESTA EQUILIBRADO Y NO TIENE VALORES.
- EN CASO CONTRARIO VOY A LLAMAR A LA FUNCION RECURSIVAMENTE SOBRE SU PARTE IZQUIERDA Y SOBRE SU PARTE DERECHA. SI LA ALGUNA DE LAS DOS DEVUELVE FALSE SIGNIFICA QUE EL SUBARBOL NO ESTA EQUILIBRADO POR LO TANTO EL ARBOL TAMPOCO Y DEVUELVO  $\langle \text{FALSE}, 0, 0 \rangle$ . SI AMBOS SUBARBOLES DEVUELVEN TRUE TIENE SI ~~DE~~ LA CANT. DE VALORES FALSOS DEL SUBARBOL IZQ ( $\#T_1$ ) DIFIERAN EN 1 A LO SUMO CON EL DERECHO. NUEVAMENTE SI ESTO NO SUCEDE DEVUELVO FALSO Y SINO DEVUELVO  $\langle \text{TRUE}, \# \text{ VALORES VERDADEROS DEL ARBOL}, \# \text{ VALORES FALSOS DEL ARBOL} \rangle$

Excelente todo el parcial  
un placer corregirlo,  
subilo a euba.wiki